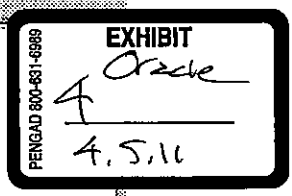
EXHIBIT 4

From: Tracey Cole Sent: 10/11/2005 8:32 PM
To: [-] Jenifer; Andy Rubin
Cc: [-] LSA
Bcc: [-]
Subject: RE: Sun meeting



thank you

From: Jenifer [mailto:jaustin@google.com]
Sent: Tuesday, October 11, 2005 7:39 PM
To: Andy Rubin
Cc: LSA; Tracey Cole
Subject: Re: Sun meeting

Larry is going to try to stop by the end of the meeting on Thurs, but he doesn't need to meet in advance to prep.

Deepest condolences,
Jenifer

On 10/11/05, Andy Rubin <arubin@google.com> wrote:

Begin forwarded message:

From: Andy Rubin < arubin@google.com>
Date: October 11, 2005 2:30:52 PM PDT
To: Larry Page < page@google.com>
Subject: Sun meeting

Larry,

We have been having discussions with Sun regarding Android's Open Source VM strategy. Alen Brenner, who owns the P&L for J2ME @ Sun is coming over to basically tell us II believe) that Sun doesn't want to work with us. Alan's big concern is that by open sourcing our J2ME VM we will make licensing "enforceability" impossible for Sun -- and he will lose revenue.

My proposal is that we take a license that specifically grants the right for us to Open Source our product. We'll pay Sun for the license and the TCK. Before we release our product to the open source community we'll make sure our JVM passes all TCK certification tests so that we don't create fragmentation. Before a product gets brought to market a manufacturer will have to be a Sun licensee, pay appropriate royalties, and pass the TCK again.

Sun has already permitted open source VM projects in non mobile areas -- areas where they didn't have a well defined revenue stream. Apache is an example.

Android is building a Java OS. We are making Java central to our solution because a) Java, as a programming language, has some advantages because it's the #1 choice for mobile development b) There exists documentation and tools c) carriers require managed code d) Java has a suitable security framework

If Sun doesn't want to work with us, we have two options:

1) Abandon our work and adopt MSFT CLR VM and C# language

- or -

2) Do Java anyway and defend our decision, perhaps making enemies along the way

As you can see, the alternatives are sub-optimal, so I'd like you to stop in our Alan meeting and essentially be the good cop. Let him know we love Sun, and want to find a way to do this Open Source thing.

I first looked to Eustace to help me, but he was out of office. Then Sergey -- out also.

Thoughts?

- andy

--
Jenifer Austin Office of the Founders. Google, Inc. office 650-253-6327.

EXHIBIT 5

From:	Brian Swetland.	Sent:1/3/2006 1:31 PM.
To: [-]	Mathias Agopian.	
Cc: [-]	fadden@google.com; arubin@google.com; joeo@google.com.	
Bcc: [-]	.	
Subject:	Re: new java world.	

[Mathias Agopian <mathias@google.com>]
> Has this decision been taken already or are we talking/arguing about it?

I think we're pretty set on it, but are still working on addressing
issues people may have with it. The skia folks are being brought up
to speed today. I unfortunately misremembered when you were going to
be back (thought it was the beginning not the end of this week) and
thought you would be around today to discuss things.

Brian

> On Jan 2, 2006, at 11:07 PM, Brian Swetland wrote:
>
> >
> >Reasons to shift to a primarily Java API
> >
> >- single language massively simplifies the application development
> > story: "you write android apps in java. native code is brought in
> > as standalone modules (services) or as plugins to the runtime
> > (components)"
> >
> >- single language approach massively simplifies system development
> > and reduces our development time. The universal multilanguage
> > binding stuff is an awesome idea but is a crazy pile of work and
> > risky.
> >
> >- the tools story is much, much simpler. supporting gcc/etc cross
> > compilers and other tools is a big pain (we have to do it for
> > systems development but we can avoid passing that pain on to
> > 99% of application developers).
> >
> >- even on a system with processes / mmu / etc, java provides a nice
> > safetynet and faster app development and debuggability. (this
> > is based on experience developing hiptop -- java saved us a
> > pretty crazy amount of time).
> >
> >- the negotiations with Sun are going far better than expected.
> > A lot of the push for multi-language bindings was a result of
> > Brian trying to work out a cover-our-ass setup for when Sun
> > proves impossible to work with (he was perhaps a bit scarred
> > by his danger experience).
> >
> >- using java simplifies the "why did you invent a new api" story.
> > The embedded java world has midp which we will have for
> >compatibility
> > with 'legacy' phone apps, but not much in the way of more powerful
> > environments. We can fill this gap *and* avoid a lot of the
> > "why didn't you just use gtk / qtopia / etc" questions.
> >
> >- using java allows us to take advantage of a modern, garbage
> > collected memory model for applications without having to worry
> > about integration with C++ allocation, reference counting, etc.
> > The same goes for synchronization.
> >
> >- having one primary language environment allows us to focus all our

> > language systems development energy on making the java world
> >extremely
> > fast and solid.
> >
> >One language is okay, but why Java instead of C++, Intercal, etc?
> >
> >- The nature of the cellular market is that we are *required* to have
> > java due to carrier requirements, etc. Since we're sorta stick with
> > that, we can provide two environments (java and native) like
> >everyone
> > else does, expending a lot more energy, or simplify and just provide
> > a fantastic java environment.
> >
> >- Java is more accessable than C++. There are more Java programmers.
> > There is more standardization in tools and libraries. Debugging is
> > much simpler (especially for people who are not total rockstars --
> > perhaps a lot of casual developers, etc).
> >
> >- Java solves a lot of the portability issues C++ has. There is
> > no fragile base class problem in the sense that it exists in C++.
> > We can safely provide a modern object oriented api to third
> > party developers without the scary ABI issues involved in C++.
> > (exceptions are zero runtime cost if not throw, by design, in
> > java. garbage collection is builtin and standard. etc. etc)
> >
> >- Performance concerns? Yup, we need to do work to push the
> > heavy lifting to native (but we're already doing that!) and
> > have a care for performance and writing more of the system in
> > java does make that a little harder. We solve this by making
> > the java runtime very fast, moving what needs to be native native,
> > and being smart about writing our java code. The folks from
> > danger can explain this at length -- shipping a *fast* java based
> > system is totally doable, even on much slower hardware than we have.
> >
> >- Java does have a big win of being much more compact code than
> > native arm/thumb code.
> >
> >What changes (not all that much):
> >
> >- The biggest change is the view system becomes a java library.
> > This does involve migrating some already written code, but in the
> > end makes for much easier to use java apis if we don't restrict
> > the design to the intersection of java and c++ features.
> >
> >- Most of the low level, performance intense stuff (image libs,
> > skia, audio engine, etc) stays just as it is and gets java
> > wrappers as previously planned.
> >
> >- The browser integration mostly moves ahead as planned. It is
> > a native component represented in a java widget/view/whatever.
> > A way of having it setup and control other widgets on top of
> > (inside of?) itself will be required for forms support, etc.
> >
> >- Joe's IDL tool is still used to generate java bindings for native
> > components, but it lives in a simpler world. It is also used
> > for building c++ and java interfaces to "services" which are
> > the bigger building blocks of the system (data storage, addressbook,
> > telephony, etc)
> >
> >Components vs Services:
> >
> >- Components plug in to the core system libraries and are exposed

> > to the java world as java classes (often wrapping native code).
> > they run inside the process space of applications
> >
> >- Services run in their own process space (though one process
> > can host a number of services that are related / can share the
> > same security boundaries). Services communicate via IPC (using
> > shared memory where appropriate) and their interfaces are defined
> > by and wrappers are generated by the IDL tool.

EXHIBIT 6

From:	Andy McFadden.	Sent:4/4/2006 2:55 PM.
To: [-]	Dianne Hackborn.	
Cc: [-]	Android Engineering [. .].	
Bcc: [-]	.	
Subject:	Re: [Android-eng] Proposal: A Variant Type.	

On 4/4/06, Dianne Hackborn <hackbod@google.com> wrote:
> One of my assumptions has been that most of the implementation of
> Variant would probably be in C++, just because it is a lot easier there
> to do things like efficiently manage a heterogenous set of types. I
> agree that this could be done as a set of specialized classes (one for
> the database rows, maybe one that can handle both Event and Intent
> data), but there is so much similar about what these things want to do
> that it really seems like it would be better for them to use a common
> facility.

Code in Java (and JS, C#, Python, etc) isn't going to be as fast as
C++, no matter what Sun is selling. However, for a user navigating
his way through an address book, it's not going to matter.

Avoid writing code in C++. The fact that we have any C++ is a nod to
efficiency concerns in very specific situations (like blitters), where
getting reasonable performance out of Java is difficult or impossible.
We will ship a more stable product sooner if we do as much as
possible in Java. We can optimize things when we know what we need to
optimize.

Sometimes you have to grit your teeth. I wrote something very like
the "event log" system described in the design docs using C#. All I
wanted to do was pull strings out of a binary stream, but I had to
jump through hoops because bytes and characters are not the same, and
you have to go through character encoding APIs to get at anything. It
was painfully inefficient. On the flip side, development took a
fraction of the time that a C++ implementation would have.

> Putting this stuff together also gives you a very powerful way of
> constructing complex data structures that can be transported to
> different environments. In fact, the kinds of structures this Variant
> would give you are exactly the same as what is provided for the JSON
> stuff (and at least a subset of the basic types could be read and
> written with the JSON text representation). It just provides it in a
> way that is more appropriate for what we are trying to do: efficient
> storage and access for basic types, easy transport across processes, and
> support for IBinder for transporting object references across processes.

Our primary goal is to ship a product, not develop astounding
technology in a handset operating system. Google has lots of research
scientists and technology people. The execs brought us in because we
have a product focus, which is very different. I would prefer not to
fail them.

At this point we need to be focused on the things we have to have to
ship. If we have to have it, it goes in; if we can live without it
for 1.0, it doesn't. If there's a simple, built-in Java way of doing
things that works the way Java developers expect, favor that over
inventing a new approach. Premature optimization is the root of all
evil.

In short, if you can do this simply and cleanly with basic Java, do
it, and move on to the next thing. There is a lot of pressure on us
to deliver, and we are falling behind.

- Andy

_____
Android-eng mailing list
Android-eng@google.com
https://mailman.corp.google.com/mailman/listinfo/android-eng

EXHIBIT 7

From:	Patrik Reali.	Sent:8/9/2005 4:51 AM.
To: [-]	Brian Swetland.	
Cc: [-]	Sascha Brawer; arubin@google.com; nicksears@google.com; miner@google.com; cwhite@google.com; fadden@google.com; tcole@google.com; ficus@google.com; Urs Hoelzle; Robert Griesemer.	
Bcc: [-]	.	
Subject:	Re: Java VM for Android.	

IMHO, another interesting project is JNode [1], a complete JVM running on the bare metal with just a tin layer of assembly. AFAIK, it has a linker for external files, which may fit your needs. The current version is for x386, but the (really nice) design makes it unaware of this and should allow for a rather simple retargeting of the whole projects.

JNode shows that you can use java for everything without need of using another system or language and still get a good performance, in part because the system is simpler and you can avoid loss due to design choices made to fit more than one system (e.g running linux + java at the same time)

On my side, I was involved in the Jaos JVM [2] (now dead), GNU Classpath [3], and I wrote a most of the simple optimizing compiler (L1) for JNode which is now the default compiler.

If you want to have an open-source java platform, you should really consider Classpath, as the project has an enourmous experience in the java libraries and regroups all the JVM developers that are done with the JVM implementation and are now improving the libraries (not to mention that getting the libraries and testsuite right is at least 10 times more work than the VM). They currently aim for J2SE but it would be great if you would help them to build a J2ME version of the classes.

Classpath is licensed in GPL+Exception, which allows you to ship the libraries in a LGPL fashion. Classpath has also the advantage that you can use it almost right away (it is very portable) and then decide which pieces to optimize (let me guess: String....).

If you decide to use Classpath or JNode, please let me know!

-Patrik

[1] http://jnode.org/
[2] http://bluebottle.ethz.ch/jaos/
[3] http://www.classpath.org/

On 8/5/05, Brian Swetland <swetland@google.com> wrote:

I am a somewhat familiar with the grungy work involved in embedded
JVM building -- I wrote the VM that the Danger Hiptop platform uses.

There are some useful reasons (in my mind) for going through the
effort of building our own embedded VM rather than just going with off
the shelf solutions:

- We'd like things to be really well-integrated with the environment,
small, fast, and fast to launch. Both the "run the vm in a little box
just for midlets" model (used by most handsets today) and "run the
entire world inside one vm" model (used by danger) have downsides.
I'd like to take advantage of running Linux on CPUs with a MMU (arm9
and better) and having multiple instances of the VM run in their own
process space. Being able to have hard limits imposed by the kernel
on memory use, etc, and tear down a whole VM if an app misbehaves is
something we wished for often at Danger. To do this, we need to make
sure we can start up things quickly when apps launch -- if java is
core to the system and not a little novelty like in current handsets,
users are not going to want to wait 10-15 seconds for apps to launch.

- License choice is important. One of the goals of this project is to provide an open source system that's appealing to handset OEMs. The Linux kernel is GPL'd, but all the pieces above the kernel that we're using or building so far are under much friendlier licenses (BSD or MIT style typically). Bringing in third party commercial solutions is tricky for this reason too, unless we plan on buying them outright or otherwise convincing them to release their software under an open source license.

- After some amazingly negative experiences at Be, dealing with Cygnus C++ compiler support, I would have a lot of concerns about throwing money (away) at Redhat or Cygnus or the like for language or compiler support. Of course there's also the concern of shopping core parts of the system out to possibly disinterested third parties.

- The JVM is going to be a central piece of the system we're building, not some little add-on on the side -- so we can provide some really good java application development and user experiences. I'd like to take recycle bits where possible to support javascript and other language bindings, which will require doing things a little differently than an off the shelf JVM.

- Classpath is interesting for their "build it all in java" approach, but from a performance perspective (which matters a lot on small devices), pushing chunks of the core library to native code is a huge win. Also, it is GPL with some special riders (which I thought the GPL disallowed...).

Anyway, those are just some points off the top of my head,

Brian

On 8/5/05, Sascha Brawer <sascha@google.com> wrote:
> Hi androids,
>
> I happened to stumble upon your wiki page [1]. Are you really sure you
> want to write your own JVM, as [2] seems to indicate? You certainly
> have your reasons, but it sounds like repeating lots of grungy work.
>
> So, if you don't mind, let me emit some random personal notes about
> the free Java scene.
>
> Everyone and their dog (not really, but way too many people) has been
> writing a JVM around GNU Classpath [3]. Most of them don't target
> embedded systems, many are crap, much has gone to oblivion, but
> there's also some stuff that might possibly be useful to you guys.
>
> I'd really recommend having a look at JamVM [4]: it's fast for a pure
> interpreter, with a decent and small codebase. JamVM is what most
> Classpath hackers use nowadays for development. The author seems a
> nice guy, he was working on optimizing Sun's and IBM's JVMs, and is
> now an independent contractor.
>
> I know of two companies using Classpath for JVMs that target embedded systems:
>
> /k/ [5] is a cover-up for one guy having his fun. My personal
> impression from the Classpath meetings is that the author is really
> into free software; I'm pretty sure he would be keen on a contract for
> an open-source embedded JVM. In the meetings list, he seemed to know
> what he's talking about, but I haven't chatted that much with him.
>
> Aicas [6] is a real company whose embedded JVM is based on Classpath.

> Since they haven't given anything back to the project, I'd be
> surprised if they would be interested in a contract for an open-source
> embedded JVM.
>
> There's also a Bytecode-to-C compiler [7], but I've no idea whether
> it's any good. I don't think it gets much used. But it might be
> interesting in case you want to use an ahead-of-time compiler for
> selected hot spots, and use a pure interpreter like JamVM for the
> rest.
>
> Kaffe [8] has been used for embedded systems, but the licensing is in
> a limbo (GPL, copyright held by a dead company).
>
> You could pay Redhat for tweaking gcj/gcc, but their focus really is
> on desktop systems. But since you mention C++ linkage on your wiki:
> gcc uses the same vtables for Java and C++, they call this "Cygnus
> Native Interface (CNI)". There has been lots of talk about giving gcc
> a better jitter for dynamically loaded bytecode; they currently have a
> very inefficient interpreter as part of the Java runtime library. But
> last I've heard, Redhat's plan now is to use the gcc backend as a JIT
> -- hairy stuff, and certainly totally unusable for an embedded sytem.
>
> If you need more info around the free Java projects, or if want to
> establish a contact, please feel to talk to either me or Patrik Reali.
> We've both been somewhat active in this scene before joining Google,
> so we know most people from meetings.
>
> Oh, you surely know that quite a few people at Google have a JVM
> background? For instance Robert Griesemer or Urs Hoelzle. I hope you
> don't mind that I'm taking the liberty to cc them on this post, in
> case they want to comment/shoot me down.
>
> Best wishes, and have fun with Android,
>
> -- Sascha
>
> [1] http://wiki.corp.google.com/twiki/bin/view/Main/Android
> [2] http://wiki.corp.google.com/twiki/bin/view/Main/AndroidJavaVM
> [3] http://www.gnu.org/software/classpath/
> [4] http://jamvm.sourceforge.net/
> [5] http://www.kiffer.be/k/profile.html
> [6] http://www.aicas.com/
> [7] http://jcvm.sourceforge.net/
> [8] http://www.kaffe.org/
>

EXHIBIT 8

From: Michael Fleming. Sent:8/16/2006 12:58 PM.
To: [-] Brian Swetland.
Cc: [-] Andy McFadden.
Bcc: [-] .
Subject: Re: manifesto.

> > - architecture: priority one is the device
> > - we are shipping the device, not the simulator
> > - if the device is not fast and stable we FAIL
> > - the emulator is the answer for desktop work
> > - yes, it is slow: we must make it faster
> > (the end users will never judge us by how fast the simulator was)
> > - performance is a problem NOW not LATER
> > - fadden's exception: allow it to build on linux-x86 for valgrinding
> > (brian's note: it need not be pretty here)

> Switching 100% to device/emulator will slow development.

Maybe the distinction here is: what do we support inside and what do we support outside? Obviously, we don't want to port the simulator to lots of targets. It's too much work already. Maybe we maintain the simulator on all current targets for internal use only and tell external people to use the emulator. I agree, we should have a grace period and see where we are in a few months.
> > - abandon single process support
> > - we are shipping multiprocess, not single process
> > - extra code to allow us to operate in two modes MUST GO
> > - hard to debug? we must write better tools
> > - again we must build and debug what we will ship

[fadden's concerns about the binder]

I agree that we must get rid of the single-process model. Maybe we have to phase it out with the simulator, maybe we have to port off the binder driver. I'd personally like to see the binder driver eliminated, but I don't know what the development time cost is. I'd like to see new stuff not use the binder. I think that's implicit in what you said below, too.

> As far as the multiprocess stuff goes, I'd like to see us abandon the
> "put anything in any process" concept and have a rigid set of things
> in each process.

Yes absolutely. I'd like to have the creeping local/remote transparency thing stopped in its tracks. Maybe it's useful to put it in the manifesto.

> > - platform: priority one is user experience
> > - if we do not ship a compelling experience
> > (dialer, pim, maps, whatever) we FAIL
> > - the platform must serve the apps & experience
> > - writing great apps must be simple
> > - "it is complicated because it is powerful" is a lousy answer
> > to "why is it so hard to do X"
>
> There are two customers:
> (1) The end-users who buy the phone;
> (2) Application developers who write apps.

There's two more, unfortunately

(3) Carriers, who will want customizations
(4) OEMs, who will write drivers, implement adaption layers and
customize to allow differentiation.

In a way, the order of priority is more like 4,3,1,2 because we can't
get out the gates without 4 and 3. We need to think more about the OEM
adaptation story than we do. We need to contain carrier demands, but
we'll still need to allow for them and prioritize them.

> #1 comes before #2.

(I agree here.)

> > - build on top of standard linux kernel services
> > - avoids making the kernel bigger
> > (can't remove core services: let's use em!)
> > - relies on well tested existing services
> > - in particular:
> > - unix domain sockets (dgram and stream)
> > - make use of the private linux-domain namespace
> > - shm passing using fd-over-socket
> > - rights checking using privs-over-socket
> > - avoid userland unix-ism
> > - we are an embedded system, not unix-on-a-phone
> > - do not keep state in a billion textfiles
> > - do not rely on the shell for anything besides debugging
> > - write it once
> > - never use two APIs or systems where one will do
> > - avoid excessive layering: in the long run, this overhead kills
> > - only one object model
> > - use the java object model
> > - do not build elaborate c++ object models
> > - minimal native code
> > - write as much as possible in java
> > - we are building a java based system: that decision is final
> > - java is easier to debug (modulo vm stability)
> > I'm serious here: GDB vs Java IDE Debuggers - think about it
> > - java is denser
> > - java is safer (bounds checking, perms, etc)
> > - favor C over C++ for native glue
> > - keep it light and fast
> > - keep it simple and clean
> > - smaller is faster
> > - write it in java first (barring certain special cases)
> > - we can always move java to native to make things faster
> > - threading model
> > - use as few as you can
> > - favor libevent/select for muxing
>
> Yup. FWIW, here's what the simulator shows at the home screen; note
> system threads are not shown (and the scraper thread wasn't running?).
>
> JamVM + Android[localhost:8000]
> Thread [<1 00000000> main] (Running)

> Thread [<20 00000000> ATChannel Dispatcher] (Running)
> Thread [<19 00000000> ATChannel Reader] (Running)
> Thread [<18 00000000> ModelInterpreter Handler] (Running)
> Thread [<17 00000000> ModelInterpreter] (Running)

These 4 are mine and will, by and large, be eliminated in the new
world. Two are only present for simulating radios.

> > - Java not Sun
> > - avoid excessive koolaid drinking
> > - we are building a java based system, not pushing Sun's agendas
> > - remember that we are CDC/MIDP, not J2SE
> > - allocate sparingly
> > - keep it small
> > - avoid reflection: it is very costly
> > - notes on design for embedded linux
> > - use unix domain sockets for local communication (secure!)
> > - use tcp sockets for debugging connections
> > (make sure we do not accept such connections over wireless network)
> > - use credential passing/checking to further restrict access if need be
> > - do *not* use sysV shm, sems, etc
> > - avoid using signals if at all possible
> > - use unix domain sockets to determine if a remote process went away
> > (when the far side closes the local side should close: verify)
> > - only the init service should start/stop processes
> > (expand it so it can queue up one-shot processes like vm instances)
> > - use inotify to monitor file
> > - perhaps we should always use libevent for muxing?

> We do need to be careful to not set ourselves back several months by
> discarding technology.

> We need to decide what needs to happen now and what can happen later.
> Anything that represents a change in focus or direction needs to
> happen sooner.

> Any significant change we make will disrupt the schedules.

Yeah. Well, for some things in the system, we can can keep using them
and slowly replace them as we get to it. I think the principles above
should be applied to all new work, though. Some stuff we have that's
non-optimal can probably be kept through 1.0, we just want to make
sure we don't get 3rd parties depending on it.

I think there's a lot of frustration over the amount of
binge-and-purge we've had on this project, particularly by those who
have been purged. Maybe the binge-and-purge can be reduced if we keep
newer stuff tighter in line with the above principles, and slowly work
to replace the old stuff rather than doing it suddenly. Part of the
problem is that we're a big enough group that we're not all thinking
the same way. People need to be encouraged to play along better. I
hate to say it, but maybe more frequent meetings and more careful
oversight by you can keep the ship on a tighter course.

I know you'd hate this Brian, but perhaps you (and Mike Cleron) should
have people sketch out design notes on subsystems and review them.

Mike

EXHIBIT 9

From:	Andy Rubin.	Sent:4/13/2006 3:41 PM.
To: [-]	Dan Bornstein.	
Cc: [-]	Steve Horowitz.	
Bcc: [-]	.	
Subject:	Re: What are we doing?.	

Obvious question - please define:

"the currently-envisioned time frame"?
and
"the currently-targeted amount"

On Apr 13, 2006, at 3:35 PM, Dan Bornstein wrote:

>> We need to provide an alternative to MSFT, and we need to do it in
>> such a way as we don't fragment 3rd party developers. See the next
>> slide in the deck for fragmentation: Java has very little
>> fragmentation, and it's adoptable. If we play our cards right, we
>> can also leverage not only existing developers, but applications as
>> well.
>
> That sounds like an argument for adopting the MIDP model for
> third-party apps, not developing a new app model.
>
>> Since we are Open Source, and the cost for our platform is close to
>> zero, we lower the total BOM by about 10%. That savings and some
>> careful choices in hardware specs helped us create our slogan:
>> "Smartphone features at featurephone prices".
>>
>> BTW, my definition of smartphone is a phone that has an open API for
>> 3rd party developers and whose own applications use that same API.
>
> Ok, even conceding the point about BOM, I do not believe we can make a
> smartphone by your definition in the currently-envisioned time frame,
> with the scope of development as currently outlined in the PRD, and
> have it be one that can be good enough to sell the currently-targetted
> amount. In the current nominal plan, there is too much that is new to
> expect everything to totally fall into place in a 1.0 release.
> Something still has to give.
>
>> I've cc'd android-team.
>
> I think you forgot to; please go ahead.
>
> -dan